Q1. What is the difference between \_\_getattr\_\_ and \_\_getattribute\_\_?

Python will call \_\_getattr\_\_ method whenever you request an attribute that hasn't already been defined. In the following example my class Count has no \_\_getattr\_\_ method. Now in main when I try to access both obj1.mymin and obj1.mymax attributes everything works fine. But when I try to access obj1.mycurrent attribute -- Python gives me AttributeError: 'Count' object has no attribute 'mycurrent'

class Count():

def \_\_init\_\_(self,mymin,mymax):

self.mymin=mymin

self.mymax=mymax

obj1 = Count(1,10)

print(obj1.mymin)

print(obj1.mymax)

print(obj1.mycurrent) --> AttributeError: 'Count' object has no attribute 'mycurrent'

If you have \_\_getattribute\_\_ method in your class, python invokes this method for every attribute regardless whether it exists or not. So why do we need \_\_getattribute\_\_ method? One good reason is that you can prevent access to attributes and make them more secure as shown in the following example.

Whenever someone try to access my attributes that starts with substring **'cur'** python raises AttributeError exception. Otherwise it returns that attribute.

class Count:

def \_\_init\_\_(self,mymin,mymax):

self.mymin=mymin

self.mymax=mymax

self.current=None

def \_\_getattribute\_\_(self, item):

if item.startswith('cur'):

raise AttributeError

return object.\_\_getattribute\_\_(self,item)

# or you can use ---return super().\_\_getattribute\_\_(item)

obj1 = Count(1,10)

print(obj1.mymin)

print(obj1.mymax)

print(obj1.current)

Q2. What is the difference between properties and descriptors?

* The most accessible technique is to use the property function to define get, set and delete methods associated with an attribute name. The property function builds descriptors for you. We'll look at this in [the section called “Properties”](https://www.linuxtopia.org/online_books/programming_books/python_programming/python_ch25s03.html).
* A slightly less accessible, but more extensible and reusable technique is to define descriptor classes yourself. This allows you considerable flexibility. You do this by creating a class which defines get, set and delete methods, and you associate your descriptor class with an attribute name. We'll look at this in [the section called “Descriptors”](https://www.linuxtopia.org/online_books/programming_books/python_programming/python_ch25s02.html).

Q3. What are the key differences in functionality between \_\_getattr\_\_ and \_\_getattribute\_\_, as well as properties and descriptors?

\_\_getattribute\_\_ is the hook that enables property (and other descriptors) to work in the first place and is called for **all** attribute access on an object. Consider it a lower-level API when a property or even a custom descriptor is not enough for your needs. \_\_getattr\_\_ is called by \_\_getattribute\_\_ when no attribute has been located through other means, as a fallback.

Use property for dynamic attributes with a fixed name, \_\_getattr\_\_ for attributes of a more dynamic nature (e.g. a series of attributes that map to values in an algorithmic manner).

Descriptors are used when you need to bind arbitrary objects to an instance. When you need to replace method objects with something more advanced for example; a recent example is a [class-based decorator wrapping methods](https://stackoverflow.com/questions/22545339/callable-object-decorator-applied-to-method-doesnt-get-self-argument-on-input) that needed to support additional attributes and methods on the method object. Generally, when you are still thinking in terms of scalar attributes, you don't need descriptors.